Finding Map Feature Correspondences in Heterogeneous Geospatial Datasets

Abhilshit Soni
Applied AI & ML Group
HERE Global B.V.
Mumbai MH IN
abhilshit@gmail.com

Sanjay Boddhu
Applied AI & ML Group
HERE Global B.V.
Chicago IL USA
sanjay.boddhu@gmail.com

ABSTRACT

In an automated map making process, map features like lane-markings, traffic-signs, poles, stop-lines and similar other features are extracted using deep learning methods from various sources of imagery or sensor data. These sources come with their own positional errors due to which the map features extracted from these sources are always misaligned with respect to each other, making the conflation of map features a difficult task. We propose a novel method to find map feature correspondences between 2 sets of map feature datasets obtained from different sources by first converting them into a heterogeneous geospatial graph and then doing node representation learning using a graph neural network that can generate vector embeddings that encode information of morphology, attributes, and absolute and relative positions of the map feature with respect to its neighbours along with aggregated information from its neighbours. This process can be employed to generate embeddings of map feature nodes, which are amicable to identifying spatially similar and corresponding map feature nodes across disparate sources with varying degree of similarity scores. When applied aptly, these map feature correspondences between two sources can be used as anchor points to perform spatial alignment with linear or non-linear transforms, leading to a better conflation.

CCS CONCEPTS

Information systems → Information systems applications → Data mining • Information systems → Information systems applications → Spatial-temporal systems → Geographic information systems • Mathematics of computing → Discrete mathematics → Graph theory → Graph algorithms

GeoKG '22, November 1, 2022, Seattle, WA, USA

ACM ISBN 978-1-4503-9538-0/22/11…$15.00
https://doi.org/10.1145/3557990.3567590

ACM Reference format:

Abhilshit Soni and Sanjay Boddhu 2022. Finding Map Feature Correspondences in Heterogeneous Geospatial Datasets. In *Proceedings of 1st ACM SIGSPATIAL International Workshop on Geospatial Knowledge Graphs (GeoKG 2022) November 1st, 2022, Seattle, USA. ACM, New York, NY, USA, 10 pages.* https://doi.org/10.1145/3557990.3567590

KEYWORDS

Map Fusion, Conflation, Alignment, Geospatial Graph

1 Introduction

Over the years map making companies have evolved to automate the process of map making which was erstwhile manual and a tedious job. A modern-day HD map usually contains accurate information of geometric map features including, but not limited to, road geometries, lane geometries, traffic signs, stop-lines, crosswalks, road surface markings etc. As part of automated map making process, such features are extracted automatically using deep learning algorithms from various sources of imagery and sensor data. These sources are usually street imagery, car sensor data or high-resolution aerial or satellite imagery, terrestrial or aerial LiDAR data etc. The data extracted from such sources come with their own positional errors that are caused mainly due to quality and configuration of sensors and the methods used to record and post process the captured inputs. Due to this, the features observed/extracted from these sources are usually misaligned with respect to each other. Also, the data obtained using deep learning algorithms when applied on a single source is a lot of times incomplete due to the nature of the source itself. For example., Aerial imagery may have occlusions like tree cover preventing the complete detection of lane markings or road boundaries. Similarly, in street imagery captured by a car with a camera mounted on the top or a dashcam can have temporary occlusions like a big vehicle in the side or front that may prevent the view of road surface or a traffic sign. Hence, map making companies usually rely on data obtained from multiple sources to build maps.

In a manual or semi-automated map making process a human operator creates or reviews map data from multiple sources, and then creates a final version of map data by fusing data from

multiple data sources manually. While, if the data is assumed to be well aligned, an automated conflation process becomes an easier task which can be achieved by clustering of map features and by filling gaps in data from one source, based on the inputs from another source and calculating the resultant attributes based on consensus. However, this important step of multi-source fusion in an automated map making process, becomes a challenging task as features from these sources are misaligned spatially and may not be amicable for spatial clustering techniques. Thus, one must align data from 2 misaligned geospatial datasets so that they get closer to each other, and then execute the semantic conflation process to derive final geometry or attributes of the fused map features.

A common way of doing alignment is by using manually surveyed ground control points as anchor points. These anchor points are used to estimate transforms between the dataset and ground control points. For example, if a dataset A is to be aligned with dataset B, with an input set of ground control points G. A transform is estimated between corresponding point features in dataset A with G and then A is transformed using the estimated transform. Similarly, a transform is estimated between corresponding point features in B and G and then B is transformed using the estimated transform. Thus, A and B both get aligned independently to G, and as a result A and B gets aligned with respect to each other. The ground control points are usually supplied by third party vendors who perform manual physical surveys of some key map features like traffic signal, pole, or traffic sign etc. Collecting ground control points in every area of interest is an expensive process and hence not scalable for those regions where survey capabilities are not available.

An alternate approach to do alignment is to estimate feature correspondences between dataset A and dataset B directly and use the matching features as anchor points or control points to estimate a transform of one dataset with respect to the another. Using the estimated transform one of the datasets is transformed so that it aligns to another dataset. The transforms estimated in both the approaches can be linear or non-linear depending on the nature of the data. The focus of this paper is on this approach of the alignment as it involves identifying inter-source map feature correspondences in form of anchor points which are precursors to estimate transforms of one dataset with respect to another. The key contributions of this paper are summarized as given below.

- To the best of our knowledge, we present the first method to construct a heterogeneous geospatial graph from a heterogeneous geospatial dataset that can have incomplete or noisy features and it also accounts for linear features in graph by processing them in a certain manner. This graph can be used to perform many other types of geospatial graph mining operations apart from the use case that is covered in this paper.
- Once the graph is constructed, we generate neighbor aggregated vector embeddings using a prominent inductive Graph Neural Network algorithm, GraphSAGE [1] that gives us vector embeddings corresponding to every map feature node in the graph that encodes the information about the morphology, attributes, relative and absolute position of itself with respect to its neighbors and aggregated with similar information passed from its neighbors and neighbor's neighbors.
- In this way we try to identify similar situations in 2 geospatial datasets by calculating the distance between their neighbor aggregated node (map feature) vector embeddings using a similarity metric like Manhattan Distance weighted by a proximity factor.
- We also publish our results of map feature correspondences on real dataset of map features extracted from different car sensor data, street imagery data and aerial data.

The intuition behind this algorithm can be explained with an example below. Let's assume that there is a no-right-turn traffic sign situated at a junction in a dataset A. In its neighborhood there is a traffic signal, a striped crosswalk, a stop-line, a road boundary and a solid yellow lane boundary that are existing at a certain distance, angle, and position. We create a geospatial graph of this dataset and generate neighbor aggregated embeddings of this traffic sign. The embeddings are generated in such a way that it includes information about the type and attributes of the no-right-turn sign itself, aggregated with type and attributes of other features in neighborhood, along with the distance between them, their absolute positions and angle with respect to a reference axis. If a similar scenario exists for a no-right-turn sign in dataset B their embeddings would be similar. Thus, yielding a match. If we get a significant amount of such confident matches these matches can be used as anchor points to estimate transforms to transform dataset A to B or from B to A. The heterogeneous geospatial datasets used for the work done related to this paper involves map feature types mentioned in Table 1.1 along with their geometric representations.

Map Feature Type	**Geometric Representation**	**Subtypes / Attributes**
Lane Markings	Polyline	Yellow, white, solid, dashed
Road Boundary	Polyline	-
Traffic Sign	Point	Speed limit, restricted driving, silent zone, hazard etc.
Road Barrier	Polyline	Guardrail, Jersey Barrier etc.
Pole	Point	-
StopLine	Polyline	-
Crosswalk	Polyline	Striped, Solid
Road Surface Marking	Point	Direction Arrows, bicycle/bus lane etc.
Traffic Signal	Point	-

Table 1.1 Map Feature Types & geometric representations

More map feature types can also be added if the data of the same is available from sources for which we want to find the map feature correspondences.

The rest of this paper is organized as follows. In section 2, we report the important related work and existing approaches used to solve similar problems. In section 3, we discuss a common step of scaling and normalization of data applicable to both linear and point features. In section 4 we describe the data preparation step to be performed for linear map features. Section 5 describes the data preparation step for point features. In section 6 we describe the algorithm to prepare the geospatial graph. In section 7 we describe the method to generate the node representations i.e., neighbor aggregated vector embeddings corresponding to each map feature in the geospatial datasets. Section 8 describes the similarity metric used to calculate distance between vector embeddings of map features in 2 geospatial datasets to identify a match. Section 9 describes the experimental setup used for using this method on real world datasets and the results obtained on the same. Section 10 describes the conclusion and future scope of this work which is followed by acknowledgements and references.

2 Review of existing work

A lot of research work has been done to conflate map data from 2 sources. To the best of our knowledge all existing work deal with conflation of geospatial datasets where map features are conflated in isolation without confirming their presence in conjunction with their neighbors. Validation of presence of map feature in conjunction with respect to its neighbors becomes important if the datasets have false-positives, which is usually the case when data is obtained through an automated map data extraction process. Such false positives should not be accounted for, while estimating the transforms between 2 datasets. With the evolution of deep learning algorithms, advances in automated map data extraction have only been possible recently. Hence, even though this topic has been worked upon over several years, this work becomes different from the other work as aligning geospatial datasets by converting them into a heterogeneous geospatial graph has not been done earlier. Since there are huge number of papers which are related to topic of conflation of geospatial datasets, it is not possible to discuss all of them in this section. Hence, those relevant to this work have been explained which includes papers related to conflation and alignment algorithms, graph related algorithms, deep learning algorithms and autoencoders which forms the basis of the work done as part of this paper.

(Gabay, 1994) [4] introduced a method to match the corresponding polylines between two different maps defined in different locations and topological characteristics. This was used for combining maps from several sources into a uniform database without geometrical and/or topological contradictions. Based on the identified set of matching entities from the different maps, they presented an automatic approach to correct and adjust the polylines from one map in order to make their locations more accurate, relative to another map. (Filin and Doytsher 1999) [20] proposed a method of matching road junctions using a point matching algorithm. Then the topology of polylines connected to the points were used to propagate matches to the lines. (Gabay and Doytsher 2000) [21] proposed a polyline conflation algorithm where polylines from one dataset are buffered and if the polyline from another source falls completely within the polygon created by the buffer the 2 polylines are considered to be matching pairs. A lot of work has been done related to aligning road network from a map data source to another road network from another map data source [5, 7, 18] or aligning road network to a satellite or aerial rasters [6], and also for aligning point clouds using point correspondences and ICP [10]. (Deretsky, 1993) [11] used a chain of arcs technique modelling the attribute and geometry for every feature in form of arcs, so that given 2 maps both maps become partitioned by the chains into much smaller pairwise matched areas called wards. The conflated map is then produced by starting with one of the input maps and attribute information from matched chain and wards is transferred to the conflated map according to a set of user configurable rules. (Dongcai, 2013) [13] defined a framework of vector spatial data conflation in multi-source vector space and discussed the flow of the conflation of attribute data and geometry data. They had identified prominent matching methods as geometric matching and semantic matching. They also describe a deterministic algorithm which can be used for feature matching based on linear feature and point feature matching based on their respective topologies.

(Doytsher 2013) [19] presented an algorithm for ad-hoc integration of road vector data, where roads are represented as polylines, but their algorithm uses only the endpoints of polylines and not the whole polylines.

(Heimann 2018) came up with The REGAL [17] or Representation based Graph Alignment Framework which was the first framework to formulate unsupervised graph alignment problem as a problem of learning and matching node representations that generalize to multiple graphs. It was defined as a generic approach that can be applied to any type of graphs. (Faerman, 2019) [18] used road junction point features to first generate a road junction node graph using the road network obtained from OpenStreetMap and a proprietary map data provider and uses a graph neural network to do node representation learning on it. The road junction nodes are then matched based on the similarity score calculated by calculating distance between their embeddings.

(Goyal, 2017) [14] summarized all node, graph and edge embedding methods in a survey of papers. (Kipf, 2017) [16] Introduced the Graph Convolutional Neural Networks (GCN) and (Bronstein, 2017) [9] introduced the idea of geometric deep learning on networks of data which laid the foundation of learning on data which is in represented in form of graph or networks. (Liu, 2020) [2] introduced the idea of graph neural networks that can be used to find neighbour aggregated and learned representations of graph components like edges and vertices by accumulating information from its neighbours. While

Node2Vec [15] introduced the idea of generating node vector representations for a graph. Node2Vec was not an inductive method, which means it does not work on unseen nodes of the graph that can be added in the future. GraphSAGE [1] was proposed as an inductive learning method for graph node and edge representation learning. (Zhou, 2020) [3] presented a review of methods in graph neural networks which also details gathering data for unsupervised learning on graphs, which was a good reference point to start for our problem statement as this involves unsupervised graph node representation learning.

(Yao 2017) [8] described a trajectory clustering method which uses LSTM [12] to generate latent representations of trajectory polylines for clustering similar trajectories.

3 Data scaling and normalization

The data extracted from different source may arrive randomly based on how it is ingested into the alignment and conflation system as an input. Hence, we must decide a unit of work on which we need to match data from 2 different geospatial datasets that needs to be merged. This unit of work can be a common or a proprietary tiling scheme that defines all data falling within that tile to be fused or conflated after performing the alignment. For our work we used HERE's tiling scheme, however any prominent tiling scheme as a unit of work can be used to replicate work of this paper. A tile is just a bounding box for which you have obtained geospatial datasets from 2 different sources between whom we need to find map feature correspondences. This bounding box tile will have a minX, minY, maxX and maxY representing the lower left and upper right corners, using which all other corners of the tile can be derived. It is easier to work with meter-based coordinates hence we first converted all the latitude longitude values representing linear and point features which were in WGS84 EPSG:4326 to their corresponding UTM zone projected coordinates. These UTM projected coordinates were then scaled in a fixed range of 0 to 10 using (minX, minY) as (0,0) and (maxX, maxY) as (10,10). In theory any other smaller range should also yield similar results.

The attributes of each linear and point feature were normalized to prepare initial attribute feature vectors. The attribute vector is a standard vector with numerically encoded or enumerated values of all type of attributes existing across linear and point features. For example, a sample attribute vector may look like enumerated version of following

$$\begin{bmatrix} sign\ type \\ sign\ value \\ lane\ type \\ lane\ color \\ lane\ style \\ crosswalk\ style \\ road\ surface\ marking\ type \\ road\ surface\ marking\ value \\ barrier\_type \end{bmatrix}$$

Where, applicable values are populated based on normalized enumerated values.

For example, a Sign type may correspond to Speed Limit Sign which can have an enumerated value of 1 and a restricted driving sign may have an enumerated value of 2 and likewise. If the feature is a Sign, the attribute vector will have non applicable values set to a default value of 0. For example, lane color is not applicable to a Sign type map feature, hence it would be populated as 0 which depicts an enumerated value corresponding to "Not Applicable". Features like road boundary, stop line, pole and traffic signal did not have any associated attributes or subtype categorization in our data hence for them the attribute vectors would be filled up with all zeros. However, when we construct the geospatial graph the node types are populated based on the feature types, so, the type information of these features is captured over there.

4 Data preparation for linear features

Linear map features like lane markings, road boundary, barrier geometries etc. are represented by a polyline. In different geospatial datasets that are obtained from different sources, same linear features can have different sampling rates of shape points. Which means that if a linear feature like a lane boundary is 100 meters long and is present in datasets A and B the polyline representing same lane boundary were represented with unequal number of shape points in both the datasets. To deal with this we decided to perform following steps to pre-process such linear features of each of the datasets.

Point features like Poles can be used to localize where we are on an adjacent linear feature. Poles are usually on the sides of road and are also on the center divider on a bi-directional road. They are usually placed at a regular interval, hence they become a good candidate point features to localize where we are on a linear feature like a road boundary, road barrier or a lane boundary. Signs, could also be used, but Signs may or may not be present on every road and also a lot of times multiple signs are posted on the same sign post or overhead banner, this could be a problem as there are chances that an automated map data extraction system might not be able to capture all of them correctly since there are multiple different sign types observed at same location in such a scenario. Hence, we decided to use Poles only for this work. We buffer each polyline laterally based on a threshold of 3 meters and check if there are any poles in that buffer. If poles are found then perpendicularly project them onto linear features to break them into multiple polylines. Threshold of 3 meters is taken considering worst-case scenario of a single lane road, with parallel poles on both sides of the road. In such a case the road boundary on each side of the road should only get cut by the pole on that side and not by the pole on the opposite side to avoid double cuts. If the number of lanes is known upfront, this threshold can be appropriately configured.

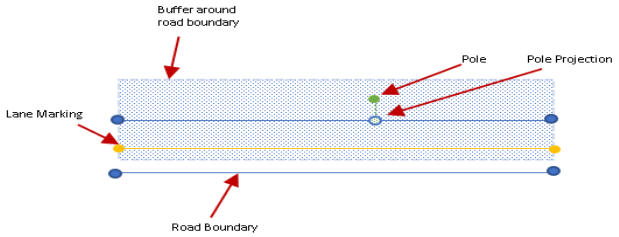

Figure 4.a Method of cutting linear features based on perpendicular projection of point features in lateral buffer

In the scenario presented in Figure 4.a, a road has 2 road boundaries and one lane boundary depicted by yellow solid lane marking. There is also a pole in the 3 meters buffer of one of the road boundaries and hence the pole is perpendicularly projected onto that road boundary and the polyline representing the road boundary will be broken into 2 polylines Start ⟶ Projected_Pole and Projected_Pole ⟶ End.

Cutting the linear feature in such a way increases the chances of finding a greater number of confident matches in the map feature matching process. The intuition behind doing this operation is that if in the above case if a confident match is found for the pole, there would be a confident match found for the linear patches of road boundary also that are created after perpendicularly projecting the pole onto it.

The second step of this process is to encode the polylines into a fixed 128-dimensional vector using a generative autoencoder like LSTM [12]. In earlier works LSTM Autoencoders have been used to encode linear features like vehicle trajectories [8] and their learned latent representations have been used to find similar trajectories by clustering them based on their embeddings. We use the same concept to encode polylines. Only difference between trajectories and polyline features like road boundaries or lane boundaries is that trajectories always have an associated direction of travel information, and their coordinates are always in order of the direction of travel. However, the lane boundary or road boundary extracted from certain sources may or may not have direction of travel information and their order of coordinates may not represent the actual direction of travel for lanes or roads. Lanes can go in opposite directions on a bidirectional road and when they are detected from aerial imagery or street imagery the order of coordinates of their geometry that are derived may not represent the actual direction of travel applicable for that lane. So, it may be possible that a lane boundary that goes from point A to point B is represented in another dataset in the order of point B to point A. Since the order of coordinates are reversed their embeddings generated by the autoencoder will not be same. LSTM encodes sequence of coordinates in their given order, if the order of coordinates are different, then their learned embeddings are also different . So, to deal with this issue we first modify the order of the coordinates of polylines so that they stay uniform in both the data sets.

To achieve this, we first iterate over every polyline and compare their start point S of the coordinate sequence with the end point E of the coordinate sequence. We first compare whether latitude of S < latitude of E if yes then we keep the order of coordinate sequence from S to E, else we reverse the coordinate sequence so that it becomes a polyline starting from E and ending at S. In case where the latitudes of S and E are equal, we compare the longitude by checking if longitude of S<longitude of E, if yes, then we keep the order of coordinate sequence from S⟶E, else we change it by reversing it so that it becomes E⟶S. This reordering is only needed if the order of coordinates in both datasets to be fused are not uniform. Once the reordering is done, we encode the reordered polylines using a polyline autoencoder which is a generative LSTM autoencoder described in the section 4.1

4.1 Polyline Autoencoder

Given that same polyline features may have different sampling rates, a generative LSTM autoencoder is capable of learning the latent representation of geometric shape and structure i.e., morphology and position represented by a sequence of coordinates irrespective of its sampling rate. For example, consider that dataset A, B and C are having the same lane boundary polyline extracted from 3 different sources respectively. They will have different sampling rates; however, the learned latent representation should be same for all 3 of them irrespective of sampling rate. Which means that autoencoders will learn the signal in the data represented by sequence of coordinates representing a polyline by discarding the noise added by different sampling rates.

To achieve this goal, we used a generative LSTM autoencoder that was trained to reproduce a given input polyline itself. Once it learns to generate itself, the latent space will hold the information on how to compress a given polyline into a fixed dimensional vector irrespective of its length, shape, structure, and number of shape points. The method used to train the polyline encoder is explained below.

20000 polylines initially were selected to train this model. These polylines were resampled at 1 point per meter rate. Additional multiple sets of this original set were created by resampling at rates of 10 meters, 15 meters, 20 meters and 25 meters. Hence total 20000x5 sets = 100000 were used as training data input. The labels used for training were 20000 x 5 sets of lines resampled at a fixed sampling rate of 10 meters. Augmentation was done to add random noise in this original input by deviating the longitude and latitude values by a random offset range within 1 meter and thus final labels of 100000 polylines were obtained. This augmentation was done to ensure that a polyline which are similar or even if it has a certain offset then the autoencoder produces an embedding vector which is closer to the representation of original polyline.
We left padded the inputs with 0 as the length of different coordinates sequences representing different polylines will not be same. Same kind of padding was also done for labels. Since the

maximum count of coordinates in the largest linear feature after equidistant resampling was 2560, we decided to left pad all other features smaller than that to match that length. Hence the shape of training data was (n, 2560, 2) where n is the sample count and 2650 is length of coordinates for each sample and 2 is the number of values in each coordinate tuple i.e., x and y (which is nothing but the scaled longitude and latitude after they were converted to UTM projection)

We used a contrastive loss function instead of MSE (Mean squared Error) as used in the original paper by (D. Yao, 2017) [8]. Since contrastive loss is more suitable while you want to map vectors that model the similarity of input items. Which is what was required in our case. Contrastive loss is mathematically described as follows.

$$loss = YD^2 + (1 - Y)\max(margin - D, 0)^2$$

Here,

- Y is the value of our label,
- D is the Euclidean distance between input and label embeddings
- max is the function that gives maximum between (margin - D) and 0
- The margin defines a radius around the embedding space of a sample so that dissimilar pairs of samples only contribute to the contrastive loss function if the distance D is within the margin

The model used is a combination of 2 LSTM networks, an LSTM encoder network, and an LSTM decoder network. The model uses a masking layer to ignore the values of mask while calculating the loss. To propagate the mask across all layers, a custom LSTM bottleneck layer was implemented that copies over the mask across the RepeatVector layer and propagates it to the decoder. The model uses a left padded 2560 length sequence of coordinates as input and encodes it to 128-dimensional vector in the bottleneck layer and decodes back a 2560 length coordinate sequence representing the same polyline that was provided as input. Refer the model architecture diagram in Figure 4.b.

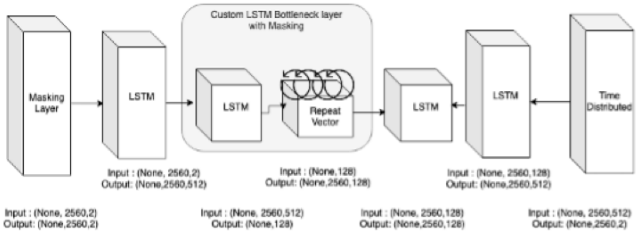

Figure 4.b Polyline Autoencoder Model Architecture

Once the model is trained, we only need the encoder part of the model to encode the polylines into 128-dimensional vector and Repeatvector and Decoder parts were pruned from the network to make the inference faster. The encoder model was then used for creating 128-dimensional vector embeddings for all linear features of the geospatial datasets for which we were to find map feature correspondences. We also prepare a scaled position vector depicting the scaled position (x, y) of the starting point of the linear feature obtained after the preprocessing and reordering as described in section 4. To incorporate the angle aspect in the feature vectors, we use the angle formed between line segment (minX, minY) ⟶ (minX, maxY) of the tile, to line segment (minX, minY) ⟶ (x, y) and append it position vector like

$$\begin{bmatrix} scaled\ x\ of\ start\ of\ polyline \\ scaled\ y\ of\ start\ of\ polyline \\ angle\ with\ reference\ axis \end{bmatrix}$$

which in turn is appended to the attribute vector and result of which is again appended with 128-dimensional embedding vector as obtained above. The resultant feature vector looks like,

$$\begin{bmatrix} \dots \\ 128\ dimensions\ of\ encoded\ polyline \\ \dots \\ sign\ value \\ lane\ type \\ lane\ color \\ lane\ style \\ crosswalk\ style \\ road\ surface\ marking\ type \\ road\ surface\ marking\ value \\ barrier\ type \\ scaled\ X\ of\ start\ of\ polyline \\ scaled\ Y\ of\ start\ of\ polyline \\ angle\ with\ reference\ axis \end{bmatrix}$$

To summarize with an example, let's assume we have 2 geospatial datasets obtained for same tile. We first normalize and scale the coordinates, prepare their attribute vectors for both point and line features, preprocess the linear features by cutting them based on point features existing in the vicinity and obtain the 128-dimensional embeddings for its resultant linear features. A position vector is prepared for each of these linear features which has scaled x and y values and angle with respect to a reference axis, which is then appended to the attribute vectors of linear features and a result of which is then appended to the 128-dimensional embeddings of the linear feature to obtain final feature vector for linear features. Thus, the final vector produced for each linear feature has information related to its morphological features, attributes and position.

5 Data preparation for point features

Point features don't have shape and structure like linear features. Hence, they can be defined just by their position vector appended to its attribute vector. However, to perform neighborhood data aggregation we must keep feature vectors of linear and point

features to be of same dimension. Hence, we used a 128-dimensional zero vector for point features appended by their attribute vector and position vector as a final feature vector for point features as shown below.

$$\begin{bmatrix} \dots \\ 128\ zeroes \\ \dots \\ sign\ value \\ lane\ type \\ lane\ color \\ lane\ style \\ crosswalk\ style \\ road\ surface\ marking\ type \\ road\ surface\ marking\ value \\ barrier\ type \\ scaled\ X\ of\ point\ feature \\ scaled\ Y\ of\ point\ feature \\ angle\ with\ reference\ axis \end{bmatrix}$$

Once the final feature vectors of linear features and point features are obtained using the method described in section 4 and section 5, they are used to produce neighbor aggregated feature embeddings using a graph neural network. But to do that we need to first construct the graph which is described in the section 6 below.

6 Geospatial Graph

Algorithm-1 describes the method that is used to build the geospatial graph. The algorithm accepts a geospatial dataset which is a collection of map features for a given tile where feature vectors are prepared as described in section 3, 4, 5 and outputs a geospatial Graph G. The steps of the algorithm are described below.

1. A KD-Tree based spatial index is constructed so that we can easily query what features are in neighborhood of a given feature in certain radius threshold of T meters.
2. An empty graph is initialized.
3. T is defined as 20 meters here so that only features within that threshold get connected to each other through an edge since they are considered neighbors.
4. Iterate over each map feature f_i in the input dataset
5. Find who is in the neighborhood within radius T for each feature f_i and store it in N_i
6. Check whether G has node f_i added already in the graph
7. If node is not in G, add it. We also populate the node type with the type of map feature we are adding as a node along with its attributes.
8. Define an empty dataframe that can hold neighboring nodes, its type, and its weights and distance from f_i.
9. Iterate over all neighbors n_j present in N_i, which are neighbors of f_i.
10. Check if these neighbor nodes are present in the graph or not
11. If node is not present in the graph, we add it along with its type and attributes

Algorithm 1: Algorithm to construct geospatial graph

Input: A geospatial dataset that is collection of map features F with their respective feature vectors V
Output: A graph G with map features as nodes along with node attribute vectors and edges created between them

Construct a KD-Tree based spatial index S using map features F
Initialize: an empty Graph G
Initialize: distance radius in meters threshold T = 20
for f_i in F: // iterate each feature in collection of map features
  N_i = S.query_radius(f_i, T) //where, N_i are neighbors of f_i in T
  if not G.has_node(f_i):
    G.add_node (f_i, type = f_i.type, attributes = $f_i.v_i$)
  Initialize: dataframe df_{ij} of node and its type, weight, distance
  for n_j in N_i: //iterate each neighbor of f_i
    if not G.has_node(n_j):
      G.add_node (n_j, type = n_j.type, attributes = $n_j.v_j$)
    if both f_i and n_j are Point features
      $\mathbf{d} = \sqrt{(f_{ix} - n_{jx})^2 - (f_{iy} - n_{jy})^2}$ // Euclidean dist
    else if one of f_i or n_j is a Polyline and one is Point feature:
      pp = perpendicular_project_point_on_line (f_i, n_j) //perpendicular projection of point feature on polyline
      $\mathbf{d} = \sqrt{(pp_{x1} - pp_{x2})^2 - (pp_{y1} - pp_{y2})^2}$
    else if both are polyline features:
      d = hausdorff_distance(f_i, n_j)
    $\mathbf{w_{ij}} = \frac{1}{1+d}$ // calculate weight
    df_{ij}.append(n_j, n_j.type, w_{ij}, d)
  end
  df_{ij} = df_{ij}.sort_by(w_{ij}, descending=True) //sort all neighboring nodes by weight
  df_{ij} = df_{ij}.unique(n_j.type, keep_first = True) // retain first unique //nodes by node type
  for unique_nearest_neighbour, w in zip ($df_{ij}.n_{ij}$, $df_{ij}.w_{ij}$)
    G.add_edge (f_i, unique_nearest_neighbour, weight = w)
  end
end
return G

12. Check If the feature f_i and its neighbor n_j are both point features
13. If both f_i and n_j are point features, then calculate the Euclidean distance between both.
14. Check if either feature f_i or its neighbor n_j is a Point feature and other one is a Line Feature.
15. If yes, then project the point feature on the line feature
16. Find perpendicular distance from Point to its projection.
17. Check if both are linear features
18. If yes, then calculate Hausdorff Distance between both.
19. Calculate weight value that is inversely proportional to D

20. Populate the dataframe with the calculated information for each node and its neighbors.
21. Sort the dataframe by its weight in descending order
22. Filter unique features based on feature type
23. Iterate over each unique node and column
24. Add edge between them. Repeat for all remaining features of input dataset.
25. Return the generated graph for the input dataset.

Since, we have 2 datasets to deal with for finding map feature correspondences amongst them, we repeat the same process for second dataset and thus prepare graph for each dataset.

7 Generating node representations

Once a graph is prepared, we need to learn the neighbour aggregated embeddings based on the neighbourhood information present in the graph. So that the neighbourhood learned embeddings will help us find matching nodes in both geospatial graph datasets.

The idea behind this can be well explained with an example of a Traffic Signal, which is in the neighbourhood of a crosswalk, a stop-line and a speed limit sign at a particular location in Graph 'A'. The same traffic signal is also present with similar neighbouring features in Graph 'B', the embeddings generated for Traffic Signal would be aggregated embeddings of itself and its neighbours, which will encompass information of shape, structure, attributes and position of itself and its neighbours. The distance between neighbourhood aggregated vector embeddings for the traffic signal in both the graph is an indicator of how confidently we can say that this Traffic Signal in Graph A is same as the Traffic Signal in Graph B. This is the reason why we need to learn neighbourhood aware embeddings.

This can be done by aggregating the embeddings of neighbours using an aggregation function. However, the concern is that this approach needs to be inductive. (Goyal, 2017) [14] presented a survey of prominent graph and node embedding generation techniques which include Matrix Factorization, Laplacian Eigenmaps, Random walk, Spectral decomposition along with other methods including the one proposed by (Grover, 2016) [15] which is Node2Vec. However, these methods are transductive which means they don't work well on unseen data or nodes which can be added into the graph in future. Using and inductive method will help speed up inference times on data belonging to other tiles which model is not trained on. (Hamilton, 2017) [1] presented GraphSAGE (Graph SAmple and AggreGatE) which can learn an inductive representation of nodes in a graph. GraphSAGE learns representation of every node by aggregating the representations of neighbours. As of today, there are many out of the box implementations of GraphSAGE available in prominent graph learning software packages. We used an existing GraphSAGE implementation as is for generating the embeddings for our graph nodes.

GraphSAGE can be trained in both supervised and unsupervised fashion. In our case we used an Unsupervised GraphSAGE which can learn embeddings of the nodes using only the graph structure and the node features, without using any known node class labels. We used a technique to generate the training data of GraphSAGE in an unsupervised manner. With this method the training samples are generated by performing uniform random walks over the graph where Positive (target, context) node pairs are extracted from the walks, and for each positive pair a corresponding negative pair (target, node) is generated by randomly sampling nodes from the degree distribution of the graph. It finally yields a positive and negative node pairs along with their respective 1/0 labels. Using this method, we generated equal number of positive and negative node pair samples from the graph for training. We used 3 random walks per node with a walk length of 7 to generate such positive, negative training data pairs.

We trained a GraphSAGE link predictor model using data of 10 level 14 HERE Tiles which were a mixture of map features obtained from 2 different sources including car sensor data from 2 separate car sensor data providers and aerial data. As a by-product of which it learns to generate the neighbour aggregated node embeddings inductively. The accuracy of link prediction obtained on a hold-out dataset was consistently greater than 70% in all our experiments, which implies that it can learn the node representations correctly.
Once the neighbour aggregated embeddings are obtained by running inference on the trained GraphSAGE model on graphs obtained from 2 separate geospatial datasets, the similarity between the nodes can be calculated by calculating distance between their embeddings.

8 Calculating node similarity

A proximity weighted Manhattan distance is used to calculate distance between node embeddings generated as part of process described in section 7. Manhattan distance is generally a good metric to use when dimensions of vectors are larger and is defined by the formula:

$$ManhattanDistance = |X1 - X2| + |Y1 - Y2| \ldots + |N1 - N2|$$

Here, X, Y, …. N are the dimensions of the vectors

We also define a proximity weight coefficient (w) as

$$w = \frac{d}{d^{(1/d)}}$$

Where, d is the distance value calculated in step 13,16 or 18 of Algorithm-1. Using both the above formulas, we define a node match confidence score function as

$$confidence_{score} = \frac{1}{1+ManhattanDistance} * w$$

9 Experimental setup and results

We conducted 2 separate experiments to test our hypothesis of finding map feature correspondence in heterogeneous geospatial graphs using node representation learning.

In our first setup we used data obtained from 3 different sources from a set of 50 level-14 HERE tiles around an urban region from Western Europe with approximately 4km^2 of area per tile which totals to 200km^2 of total area. The sources of data were, two different car manufacturers using 2 different type of camera, GPS and IMU sensors and the 3rd source was map features extracted from HERE's own street imagery dataset. For the confidentiality purposes, let's refer the two car manufacturers providing car sensor data as C1 and C2. So, C1, C2 and HERE all 3 sources had data available for those 50 tiles with all the feature types described in Table 1.1. This data was used as a hold-out dataset that was separate from the data used for training the GraphSAGE and polyline Autoencoder models which was as described in their respective sections. On analyzing the datasets from these three sources, we found that an average misalignment between any 2 pairs of sources is not more than 4 meters. Hence, we also decided to create a second experimental setup where used dataset of C1 and C2 and augmented one of C2 dataset by adding fixed offset of 10 meters to its original positions and we also dropped certain percentage of map features, and we ran our tests on both the original and the augmented dataset as we kept on dropping nodes from the augmented graph. For both the setups, we generated the neighbor aggregated embeddings using our method for all the datasets after converting them into geospatial graphs using the algorithm we have proposed. Then we measured the percentage of highly confident map features obtained between pairs of these datasets by putting a cut-off value on the match confidence score. We measured how many map features are matched correctly at different threshold values of confidence score by manually analyzing the same. Table 9.1. shows the percentage of matching features with high confidence in given source pairs.

Cut off	Source #1	Source #2	% match	% correct
>75%	C1	C2	~64 %	98 %
	HERE	C2	~57 %	99 %
	C1	HERE	~53 %	99 %
>90%	C2	C1	~42 %	100 %
	HERE	C2	~39 %	100 %
	C1	HERE	~40 %	100 %
>90%	C1	C2 with Offset 10m	~55 %	100 %
>90%	C1	C2 offset+ 10% nodes deleted	~32 %	100 %
>90%	C1	C2 offset+ 20% nodes deleted	~24 %	100 %

Table 9.1. Percentage of matching features with different confidence score cut-offs and source dataset pairs

The "% match" column indicate % of how many total nodes matched in graph of source#1 to nodes in graph of source#2. The column "% correct" is the % of correct matches out of the total found matches at that confidence cut-off score.

At the confidence score cut-off of 75% the percentage of overall matches is more than 50%, across the combination of various data sources. At this threshold all matches are highly confident matches with very minimal ~2% incorrect matches. At a cutoff of 90% for both the experiments on an average 38% of map features are being matched with very high confidence and at a precision of 100%. These map feature correspondences being accurate can be used as anchor points for alignment purposes without any hesitation.

If we lower the confidence score threshold, the match % still increase but the precision of match goes down. This happens because a similar node exists slightly far off from the position of the actual match for certain nodes which has similar kind of neighborhood. Also, once we lower the confidence score threshold, we can get one to many matches, with the closest one being the one with highest score. Some incorrect matches can be eliminated by either picking the best candidate based on confidence score if there are such one-to-many matches, while in some cases an additional distance-based filter may help. In short as long as we are okay with the number of highly confident points for our downstream alignment task, we don't need to lower the threshold further.

In ground control point (GCP) based alignment method the ground control points provided by third party usually comprise of 10% to 15% of the nodes for the entire dataset. With this method we can have more than 38% of highly confident control points, which can be used to align the datasets.

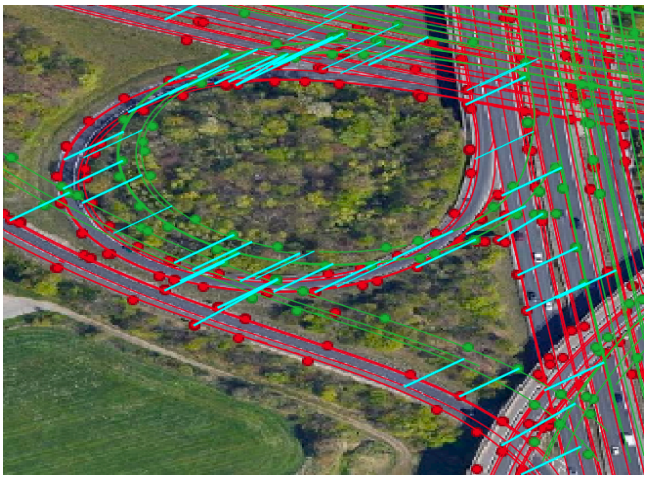

Figure 9.a. Map feature correspondences (cyan) between Car sensor data C1 (red) and Car sensor data C2 (green) augmented with 10m offset and 25% nodes removed.

In the results of experiment done with augmented dataset of C2 with C1 which has >10 meter of offset. Even after dropping a few nodes randomly, it still is able to report confident matches at 90% threshold. However, the % of confident matches reduces as we keep on removing the nodes further. For reference, Figure 9.a. shows the obtained map feature correspondences in a section of a tile which was part of our hold-out dataset.

10 Conclusion and future scope of work

In this paper, we presented a method to construct a heterogeneous geospatial graph comprising of multiple different map feature types including linear and point map features by processing them in a certain manner so that they can be used to do various types of geospatial graph mining operations.

Further, we have demonstrated how these generated geospatial graphs can be leveraged to learn the neighbor aggregated vector embeddings so that they can be used to solve automated map feature alignment problem, by finding map feature correspondences in a similar dataset obtained from a different source.

Using graph mining algorithms one can learn the usual patterns of certain map feature types, like with whom they are mostly associated with or even learn a representation of a map feature to categorize outliers by discarding them if their embeddings are very different from usual embeddings of that map feature. For example., if we cluster all learned representations of traffic signals it should give us a cluster centroid of how a usual traffic signal is represented, and even help us filter out false positive traffic signals that were reported at wrong places than the usual traffic signals with a certain set of features in its surroundings. Our future work is planned to explore this path to remove false positives by doing anomaly detections in automated map making domain.

Another area to explore could be prediction of missing or undetected attributes or features. For example., a lot of time signs are detected but they are not categorized into subtypes whether it's a speed limit sign, or a no-turn sign etc. Such missing classifications can be predicted if we create such a geospatial graph from ground truth data and use it with graph neural networks that can predict missing nodes or nodes with missing attributes.

With respect to current work discussed in this paper, we can also improve it further by supporting 3D geometries. Polyline Autoencoder can be changed to encode 3D polylines and position vectors can be modified to store the elevation value while training the GraphSAGE model.

ACKNOWLEDGMENTS

This work would not have been possible without support and funding provided by the managers and leaders at HERE Global B.V. Also, the data providers and engineering teams who provided the real-world data for us to work and experiment on. Also, we would like to acknowledge the contribution of Data Science faculty members associated with Birla Institute of Technology and Sciences, Pilani, who did review this work and provided their valuable feedback.

REFERENCES

[1] Hamilton, W. L. (2017). Inductive representation learning on large graphs. Proceedings of the 31st International Conference on Neural Information Processing Systems, (pp. 1025-1035).
[2] Liu, Z. & . (2020). Introduction to graph neural networks. Synthesis Lectures on Artificial Intelligence and Machine Learning, 1-127.
[3] Zhou, J. C. (2020). Graph neural networks: A review of methods and applications. AI Open,57-81.
[4] Gabay, Y. &. (1994). Automatic adjustment of line maps. In proceedings of GIS/LIS, 94, pp. 332-340.
[5] Zhang, M. &. (2010). A road-network matching approach guided by 'structure'. Annals of GIS,165-176.
[6] Chen, C.-c. (2005). Automatically and accurately conflating road vector data, street maps and orthoimagery. PhD. Thesis, University of Southern California, Los Angeles, CA, USA.
[7] Yang, B. &. (2012). A probabilistic relaxation approach for matching road networks. International Journal of Geographical Information Science, 1-20.
[8] D. Yao, C. Z. (2017). Trajectory clustering via deep representation learning. International Joint Conference on Neural Networks (pp. 3880-3887). IJCNN.
[9] Bronstein, M. M. (2017). Geometric Deep Learning: Going beyond Euclidean data. IEEE Signal Processing Magazine, pp. 18–42.
[10] Yue, Y. &. (2020). Point Registration Approach for Map Fusion. In Y. &. Yue, Collaborative Perception, Localization and Mapping for Autonomous Systems (pp. 29-45). Springer.
[11] Deretsky, Z. &. (1993). Automatic conflation of digital maps. Proceedings of VNIS '93 -Vehicle Navigation and Information Systems Conference. IEEE.
[12] Hochreiter, S. &. (1997). Long short-term memory. Neural Computation, pp. 1735–1780.
[13] Dongcai, H. (2013). A Study on Theory and Method of Spatial Vector Data Conflation. Research Journal of Applied Sciences, Engineering and Technology, 563-567.
[14] Goyal, P. &. (2017). Graph Embedding Techniques, Applications, and Performance: A Survey. Knowledge-Based Systems.
[15] Grover, A. &. (2016). node2vec: Scalable Feature Learning for Networks. KDD: proceedings. International Conference on Knowledge Discovery & Data Mining.
[16] Kipf, T. & W. (2017). Semi-Supervised Classification with Graph Convolutional Networks.International Conference on Learning Representations.
[17] Heimann, Mark & Shen, Haoming & Safavi, Tara & Koutra, Danai. (2018). REGAL: Representation Learning-based Graph Alignment. 117-126. 10.1145/3269206.3271788.
[18] Faerman, E., Voggenreiter, O., Borutta, F., Emrich, T., Berrendorf, M., & Schubert, M. (2019). Graph Alignment Networks with Node Matching Scores.
[19] E. Safra, Y. Kanza, Y. Sagiv & Y. Doytsher (2013) *Ad hoc* matching of vectorial road networks, International Journal of Geographical Information Science, 27:1, 114-153, DOI: 10.1080/13658816.2012.667104
[20] Filin, S., & Doytsher, Y. (2000). A Linear Conflation Approach for the Integration of Photogram-metric Information and GIS Data.
[21] Yair Gabay and Yerahmiel Doytsher. AN APPROACH TO MATCHING LINES IN PARTLY SIMILAR ENGINEERING MAPS. *GEOMATICA*. **54**(3): 297-310. https://doi.org/10.5623/geomat-2000-0042